

**《计算机算法课程实践》**

**实验报告**

**实验（ 二 ）**

**年 级： 2024**

**专 业： 计算机技术**

**姓 名： 刘兴宇**

**学 号： 2120240739**

# 1 实验内容

* 实现反馈式调整和启发式预测两种动态深度限制限制调整策略。
* 解决八数码问题。
* 说明两种策略具体设计，棋盘存储结构，搜索过程，搜索路径以及性能分析。

# 2 设计思想

**1.反馈式调整：**

采用迭代加深算法算法实现反馈式深度调整。初始深度设置为4，每次迭代最大深度加一。

DFS参数包括当前深度、最大深度、扩展节点数、path、closed、当前状态、目标状态。其中棋盘使用string形式进行存储。

首先判断当前深度与最大深度的关系，若当前深度大于最大深度则返回false，否则继续执行。

其次判断当前节点是否已在closed中，若在closed中则返回false，否则将该节点添加到closed中。

之后判断当前节点与目标状态的关系，若当前节点是目标状态则将节点添加到path中，并返回true，说明找到目标状态，开始回溯。

若当前节点不为目标状态，则找到当前节点中的空格索引，将该索引转换为二维坐标。遍历空格位置上下左右四个方向，注意此处有边界限制，越界直接跳过。将空格与其他方向上的字符交换位置得到新的节点，进行递归调用。

若递归过程中调用栈中的dfs返回true，则将该节点也添加进path中，说明该节点为路径上的一点。

**2.启发式调整：**

采用启发式的迭代加深 A\*（IDA\*）算法实现反馈式深度调整。初始估值上限设为4，每次迭代过程中根据子节点反馈的最小估值动态调整当前估值界限，以更有效地接近目标状态。

搜索函数参数包括当前节点信息（状态、代价、空格位置）、估值上限、扩展节点计数器、路径栈以及去重集合。状态使用字符串形式存储。

首先计算当前节点的估值（代价加启发值），若超过当前估值上限，则直接返回该估值，用于后续反馈调整；否则继续扩展。

接着判断当前节点是否为目标状态，若是，则将其压入路径栈并返回标志值，表示找到解，开始回溯。

若当前节点未达目标状态，则先将其加入去重集合，避免重复搜索；然后定位空格字符的位置，转换为空格所在的二维坐标。

遍历空格上下左右四个可移动方向，每个方向若越界则跳过；否则与空格交换生成新状态，并检查该状态是否已被访问。

对每一个合法新状态，递归调用搜索函数，若递归返回成功标志，则说明当前节点位于解路径上，将其也压入路径栈，继续回溯。

若所有分支均未找到解，记录过程中遇到的最小估值，并在本轮迭代结束后据此更新下一轮的估值上限，实现反馈式深度调整。

搜索过程持续进行直至找到解或达到预设的最大估值上限，最终输出解路径、扩展节点数及运行时间等信息。

# 3 程序效果

程序的运行效果，输入及输出的相关要求和具体执行结果如下所示：

1. 利用迭代加深算法解决八数码问题。

2. 利用迭代加深A算法解决八数码问题。

## 3.1 输入

算法对应的数字。1-BFS、2-IDDFS、3-IDA\*。

## 3.2 输出

运行时间、扩展节点总数、解路径长度、最大深度以及解路径。

IDDFS输出示例：

Time used: 114 ms

Total number of extended nodes: 67598

Lenth of path is 16.

Maximum depth reached: 16

Path:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 4 2  5 6  8 3 1 | 7 2  5 4 6  8 3 1 | 7 2  5 4 6  8 3 1 | 7 2 6  5 4  8 3 1 | 7 2 6  5 4  8 3 1 |
| 7 2 6  5 4  8 3 1 | 2 6  7 5 4  8 3 1 | 2 6  7 5 4  8 3 1 | 2 6  7 5 4  8 3 1 | 2 6 4  7 5  8 3 1 |
| 2 6 4  7 5  8 3 1 | 2 6 4  7 3 5  8 1 | 2 6 4  7 3 5  8 1 | 2 6 4  7 3  8 1 5 | 2 6 4  7 3  8 1 5 |
| 2 4  7 6 3  8 1 5 | 2 4  7 6 3  8 1 5 |  |  |  |

IDA\*输出示例：

Time used: 5 ms

Total number of extended nodes: 302

Length of path: 16

Maximum depth reached: 16

Path:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 4 2  5 6  8 3 1 | 7 2  5 4 6  8 3 1 | 7 2  5 4 6  8 3 1 | 7 2 6  5 4  8 3 1 | 7 2 6  5 4  8 3 1 |
| 7 2 6  5 4  8 3 1 | 2 6  7 5 4  8 3 1 | 2 6  7 5 4  8 3 1 | 2 6  7 5 4  8 3 1 | 2 6 4  7 5  8 3 1 |
| 2 6 4  7 5  8 3 1 | 2 6 4  7 3 5  8 1 | 2 6 4  7 3 5  8 1 | 2 6 4  7 3  8 1 5 | 2 6 4  7 3  8 1 5 |
| 2 4  7 6 3  8 1 5 | 2 4  7 6 3  8 1 5 |  |  |  |

# 4 算法分析

**IDDFS 算法代码分析**

时间复杂度分析

整体时间复杂度: O(b^d)，其中b是分支因子(≤4)，d是解的深度

核心DFS函数分析:

每个状态最多被访问一次(通过closed集合保证)

每个状态最多有4个后继状态(上、下、左、右移动)

探索每个状态的时间复杂度为O(1)

迭代加深过程:

第1次最大深度为4: O(b^4)

第2次最大深度为5: O(b^5)

...直到找到解为止

实际效率提升:

代码中使用closed集合避免重复访问状态，减少了实际计算量

每次迭代增加深度时，重置closed集合和路径栈

空间复杂度分析

最坏情况: O(b^d)，来自于closed集合存储所有访问过的状态

实际空间使用:

closed集合存储已访问状态，随搜索深度增加而增长

path栈: 存储解路径，空间复杂度为O(d)

递归调用栈: 深度为d，空间复杂度为O(d)

**IDA\* 算法代码分析**

时间复杂度分析

最坏情况时间复杂度: O(b^d)，其中b是分支因子(≤4)，d是解的深度

实际复杂度：比普通IDDFS更优，平均为O(b^(d/2))。

其中:

启发式函数(曼哈顿距离)大大减少了搜索空间

每次迭代仅探索f值不超过当前阈值的节点

阈值更新策略避免了无效迭代，每次确保拓展有意义的边界节点

启发式函数计算:

曼哈顿距离计算为O(n)

通过预计算目标位置(target\_pos)优化了计算效率

迭代加深过程:

每次迭代的搜索边界由启发式函数决定

可能跳过某些深度，直接到更有前景的深度值

空间复杂度分析

最坏情况: O(b^d)，由于使用了closed集合存储访问过的状态

实际空间使用:

closed集合: 存储当前搜索路径中的状态，当回溯时会删除节点

path栈: 存储解路径，空间复杂度为O(d)

递归调用栈: 深度为d，空间复杂度为O(d)

# 5 总结

IDDFS算法和IDA\*算法中在回溯过程中需要清空closed表。